Score Based Generative Modeling
Deep Learning 2022
Jules Kuperminc

The paper presents stochastic differential equations that transforms a complex data distribution by slowly injecting noise and a reverse-equation to generate new data points from noise. Combining score-based models with different solvers, it suggests a method to generate new images from existing image data sets. After quickly testing the model on another popular image data set (FashionMNIST), I explored short sound generation using a data set composed of 6549 single note sounds, generated using a digital edition software and python audio edition libraries.

Useful links

- Link to Google Colab
- Link to Google Drive

1 Summary of the paper

1.1 Score-based generative models

The paper explores score-based generative model, an alternative solution to generative adversarial network for data generation. Those techniques inject noise in the training data to generate new data points. When GANs relies on two competing neural networks to generate new points, score-based generative models offer a more flexible canvas where the initial data distribution is learned by a score-based model (with the constraint of having equal input and output dimensions). More specifically, those models offer interesting performances for image generation.
Given a probability density function $p(x)$, the score is defined as $\nabla_x log(p(x))$. The key idea of score-based models is to approximate the gradient of the log-likelihood with respect to the input dimensions, instead of computing the probability density function. Those models are easier to parameterize and do not require the use of a normalization constant. Those models have significantly less limitations than other classes of generative models (GAN, likelihood-based model).
In energy based models, the pdf $p_\theta(x)$ is expressed as $p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta}$ (where Z_θ is the normalization constant) and the score can be written $-\nabla_x f_\theta(x)$ (does not depend on Z_θ). The first step (score estimation) is to learn θ by fitting $\nabla_x f_\theta(x)$ to the data, through score-matching.
In a discrete setting, the sample generation process is done using Langevin Dynamics. The process consists in to follow the direction of the gradient we have estimated and to add noise to signal in order to explore the distribution and sample from it:

- estimate $\nabla_x log(p(x))$
- initialize x_0
- for $i = 1, \dots, T$
  - z_t $N(0, 1)$
  - $x_t = x_{t-1} + \frac{\epsilon}{2} \nabla_x log(p(x_{t-1})) + \sqrt{\epsilon} z_t$

At each step the new point is generated by adding a gradient component and a gaussian noise to the previous point.
This first version present several limitations:

- in a high dimensional data setting $\nabla_x log(p(x))$ might be undefined in some regions

- score estimation is inacurate in low-density regions
- Langevin dynamics may fail when the distribution has disconnected support

One solution is to use annealed Langevin dynamics. The idea is to sample sequentially with Langevin dynamics using $\sigma_1 < \ldots < \sigma_L$ (run a first Langevin dynamics with σ_1, then a second with σ_2 using the results from the first dynamics).

1.2 Score-Based Generative Modeling through Stochastic Differential Equations

The idea is to corrupt data using a continuous diffusion process. Reversing the diffusion process leads to sample generation.
Let $x(t)$ be a diffusion process, governed by a stochastic differential equation:

$$dx = f(x,t)dt + g(t)dw$$

Where,

- $f(x,t)$ is the drift coefficient
- $g(t)$ is the diffusion coefficient
- w is a brownian motion

Sample generation is done by reversing the equation. Starting from a distribution p_T (last noised distribution) and reversing the diffusion process, we can obtain samples from p_0 (initial distribution). Denoting **w** the reverse brownian motion, the reverse process is written:

$$dx = [f(x,t) - g^2(t)\nabla_x log(p_t(x))]dt + g(t)d\mathbf{w}$$

The paper then explores three samplers to generate samples from the data distribution.

- Numerical solvers such as the discrete Euler Maruyama solver
- Predictor corrector solvers
  - (predictor step) sample $x_{t-\Delta t}$ from x_t
  - (corrector step) refine the sample using several steps of Langevin dynamics
- Numerical ODE solvers
  - solve the ordinary differential equation that is associated to the stochastic differential equation

2 Experimental work

Introduction

Summarize the work: - Using the paper's code - 3 neural networks: different configurations for the layers and channels - Produced new samples and compared the 3 samplers - Created my own data set

2.1 Data set and data processing

The data set is composed of 6549 single-note wav files ranging from G3 to G5 (Sol3 to Sol5) with a duration of 0.37s per file.
All sounds have been generated using the digital edition software Logic Pro X and Apple's default digital intruments. Each note has been played by 59 different music instruments (different settings on the digital synthesizers) including pianos, chords, winds and electronic organs for a duration of 1.2s. Each sound is then split into 3 different 0.37 files (deleting the end of the file to cut silence). All audio edition tasks has been performed with the torchaudio, librosa and scipy.io Python libraries. All 6549 single-note file are then converted from stereo to mono and submitted to a low pass filter (with a cutting frequency of 10 kHZ).
The duration of each file has been chosen in order to guaranty a 128*128 size for the waveform tensor, with a sampling frequency of 44100 Hz.
Each tensor is reshaped into a 128 x 128 tensor to fit into the different U-Nets neural networks.
All files are stored in a distant Google Drive folder and can be loaded by uncommenting the relevant cells in the notebook.
The dataset class takes a folder as an input and

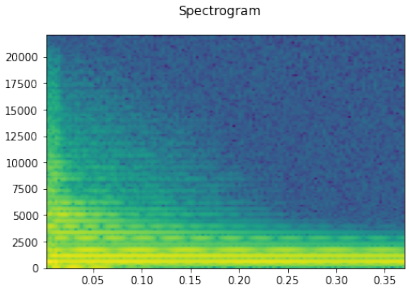

Figure 1: An example of spectogram

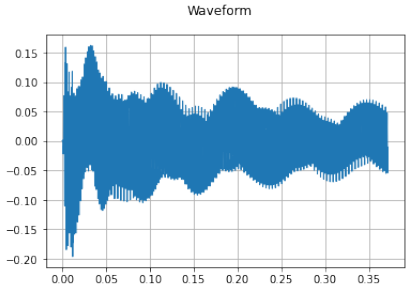

Figure 2: An example of waveform

creates a set of 128 x 128 tensors that can be used in the dataloader class.

2.2 Audio data visualisation

To visualize the initial wav files and the samples, I have created several functions to plot each sound's spectrogram and waveform (using scipy.io). The IPython.display library offers a simple audio API to listen to each file. This can be seen in the Data visualisation part of the notebook.

2.3 U-Net Neural networks

In addition to the Neural Network provided with paper (U-Net with 4 DBlocks), I adapted the existing code to test two other similar U-Nets neural networks:

- 5 DBlocks U-Net: adding conv layer and a dense layer in the encoding part and a tconv in the decoding part
- 6 DBlock U-Net: similar with two layers

Those deeper neural networks offer the opportunity to explore larger values for channels. Based on another article using score-based generative models to generate drum sounds

Figure 3: Sample images generated using the Fashion-MNIST model

(3), I decided to stick to the U-Net architecture that appears as relevant for sound generation. The U-Net used in the article (3) was more complex, with FiLM layers. This was the opportunity to explore different settings for the channel values. I tested the 4 blocks with the initial channel values: [32, 64, 128, 256].

2.4 Test on the FashionMNIST data set

In order to test the model on different data set, I initially tried to generate images using the FashionMNIST data set (dataset composed of 32 x 32 tensors with 1 channel. For this purpose, I used the initial 4 blocks U-Net network with [32, 64, 128, 256] as channel values and the Euler-Maruyama sampler. The results are displayed on figure 3.

2.5 Training and hyperparameters tuning

Training was performed using the gridtrain function to iter on several hyperparameters. The different hyperparameters were channels, learning rate, number of epochs and number of blocks in the U-Net:

- 4,5 and 6 blocks in the U-Net
- 10, 20 and 50 epochs
- 1e-3, 1e-4 and 1e-5 for the learning rate

For the 4 blocks U-Net, the values for the channels were: [32, 64, 128, 256], [128, 128, 128, 256], [128, 256, 256, 512] and [128, 256, 512, 1024] The 5 and 6 blocks layer were

trained using a learning rate of 1e-4 and 50 epochs, the chosen channel values were: 5 blocks: [128, 128, 128, 128, 128], [128, 256, 256, 512, 1024], [128, 256, 256, 256, 512], [128, 256, 512, 1024, 2048]
6 blocks: [64, 128, 256, 512, 1024, 2048], [128, 256, 512, 512, 1024, 2048], [128, 256, 512, 1024, 2048, 4096] and [128, 256, 256, 512, 512, 512]
Using the grid function, the different models were trained and samples were generated using the 3 sampler suggested in the paper.
With input data of dimension 128 x 128, the lowest channel value is 128. Models and generated samples can be loaded from the Drive.

2.6 Samplers

All samples were generated using the 3 samples suggested in the paper. The code was adapted to transform the output (128 x 128 tensor) into a new wav file. Samples were plotted and displayed at the end of the notebook.

2.7 Likelihood

Given the results, I did not computed likelihood computation. Instead, the idea was to listen to each sample and observe the visualisations to determine which were acceptable samples.
Most samples were far from actual sounds or music notes. The method has mostly created white noises or heavily noised samples that can be seen at the end of the notebook.

Conclusion

With input files that are larger (128 x 128) than the hand-written and the fashion images datasets, the results obtained with similarly shaped neural networks are not satisfying. I also performed some computations using a 1D tensor (128*128 long vector instead of a 128 x 128 matrix) without success. With more time and available computation, it would have been interesting to compare the results with a typical GAN architecture as well as with a larger range of neural networks. Hypothesis for the lack of satisfying results include:

- too diverse dataset with 36 different tones and 59 different music instruments that makes it difficult for the network to learn. However, it was difficult to build a simple dataset large enough otherwise.
- wrong neural network for this specific type of data. Article (3) adopts a slightly similar architecture but the data are simpler and more homogeneous.
- wrong transformation of waveform-like data into square matrix

A promising solution that I would like to explore later is to train the network on a set of short MIDI files were 3 X 32 x 32 tensor would represent a short melody (3 channels for the intensity or velocity of the note, each line representing a tone and each column a start time). Such approach requires a large dataset of melodies: a solution to build such a dataset is to use existing classical music MIDI files and to segment them into short samples.

References

[1] A Comparison on Data Augmentation Methods Based on Deep Learning for Audio Classification, Shengyun Wei et al 2020 J. Phys.: Conf. Ser

[2] Deep Learning Techniques for Music Generation Compound and GAN, Jean-Pierre Briot, LIP6

[3] Raw Audio Score-based Generative Modeling for Controllable High-resolution Drum Sound Synthesis, Simon Rouard, Gaëtan Hadjeres, SONY CSL

[4] Inject Noise to Remove Noise: A Deep Dive into Score-Based Generative Modeling Techniques, https://wandb.ai/

[5] Data Augmentation for Audio, https://medium.com/